

Introduction

Logics plays an essential rôle in Computer Science and in Artificial Intelligence.

This will be demonstrated with some examples in this introduction.

0
0/0

Programming

Within the last decade it turned out that computerised systems are the very base of advanced technology. Software is present in nearly all devices of modern houses, in our cars, not to speak about aircrafts or weapons. Without going into details, it is immediately obvious that for most, if not all applications, robust, safe and correct behaviour of a system is mandatory. To achieve this it is widely accepted that it is only possible if formal methods are applied during the entire process of hard- and software-development. In the following we shortly depict some tasks where the use of logic has proved to be extremely helpful.

0/1
0/1/0

Abstract Datatypes In order to define datatypes and to derive efficient implementations for it the concept of abstract datatype definitions is central. The idea is to define the abstract properties of a datatype, instead of giving special realisations. A very trivial example is the definition of a *stack*:

Let Σ be an alphabet. In order to define a stack S over Σ we assume *clear* to be a nullary function and we define the following properties of stacks:

0/1/1



Go Back

View Size

Single Page

Continuous

$$\begin{aligned}
& \text{clear} \in S \\
& \forall s \in S \forall x \in \Sigma \ (push(s, x) \in S) \\
& \forall s \in S \ (s \neq \text{clear} \Rightarrow pop(s) \in S) \\
& \forall s \in S \ (s \neq \text{clear} \Rightarrow top(s) \in \Sigma) \\
& \forall s \in S \ (empty(s) \in \{true, false\})
\end{aligned}$$

Hence, we have the functions *clear*, *push* and *pop*, which yield stacks and the predicate *empty*, furthermore we need the following properties with respect to the *push*-operation.

$$\begin{aligned}
& \forall s \in S \forall x \in \Sigma \ (push(s, x) \neq \text{clear}) \\
& \forall s \in S \forall x, y \in \Sigma \ (push(s, x) = push(s, y) \Rightarrow x = y) \\
& \forall s, t \in S \forall x \in \Sigma \ (push(s, x) = push(t, x) \Rightarrow s = t)
\end{aligned}$$

And we have to give properties with respect of the combinations of functions:

$$\begin{aligned}
& \forall s \in S \forall x \in \Sigma \ (top(push(s, x)) = x \wedge pop(push(s, x)) = s) \\
& \forall s \in S \forall x \in \Sigma \ (empty(push(s, x)) = false) \\
& \qquad \qquad \qquad empty(\text{clear}) = true
\end{aligned}$$

0/1/2

The above formulae state what properties we expect stacks to have. Obviously it contains no



Go Back

View Size

Single Page

Continuous

hints how to implement such a data type. And indeed this specification is aiming to solve other problems, e.g.

- Is the specification correct? I.e. is there a set S together with the given operations, such that the axioms above hold?
- Is the specification complete. I.e. do the axioms imply all the properties we intuitively assume a stack to have? Are there sets S which do not meet our expectations?

The reader may have already noticed, that the axioms are nothing else then formulas in predicate logic, i.e. al logic where variables like x or s togehter with so called quantifiers \forall and \exists are used.

The above two questions, namely correctness and completeness, are very central topics for the design of formal systems in logics. The prove of these proprties often is difficult and costly, but on the other hand it is one of the clear advantages of logical systems, that these properties *can be proved formally*. In the main part of this course we will deal with these questions explicitly.

0/1/3

Program development There are a number of attempts to define methods, which allow the development of a program togehter with a formal argument of its correctness. We will give a very rough idea with a toy example.

Assume the following simple program containing a loop, and assume that a denotes an array of integers:

```
max := a(1);
do i = 2,n
if a(i) > max then max := a(i)
end
```



Go Back

View Size

Single Page

Continuous

In order to understand what is going on if this program is executed, the following so called *loop invariant* is helpful

$$\forall j : 1 \leq j \leq i \Rightarrow \max \geq a(j)$$

This means, that for every value of i , i.e. before and after every execution of the if-statement within the do-loop the above formula is valid. Now assume that the loop is executed the last time, hence the value of i after executing of the loop-body is n , one can conclude that \max contains the maximum value of the array:

$$\forall j : 1 \leq j \leq n \Rightarrow \max \geq a(j)$$

0/1/4

Another important issue for program development is the **specification of programs**. In order to give a formal specification of the above program for finding the maximum of an array the following logical formula can be used.

$$\forall S \forall m \max(S, m) \Leftrightarrow (S \neq \emptyset \Rightarrow (m \in S \wedge \forall x (x \in S \Rightarrow m \geq x)))$$

Note that in this specification S is assumed to be a set. There is no decision yet made, that this has to be implemented by means of an array.



Go Back

View Size

Single Page

Continuous

On the other hand logic can be used not only for the specification but also as programs itself. The following is a logic program which computes the maximum value of a list of values. Lists are represented as `[head.tail]`, where `head` denotes the front element of a list, `tail` the rest of the list and `nil` the empty list.

```
max([m.nil], m) <- .
max([head.tail], m) <- max([tail], m),
                       head < m.
max([head.tail], head) <- max([tail], m),
                          head >= m.
```

Artificial Intelligence

One of the oldest sub-disciplines of Artificial Intelligence (AI) research is automated theorem proving. In the early days some were very optimistic about using theorem provers as general problem solvers for various different tasks, like action planning, knowledge representation or program verification. Now it is clear, that for special tasks tailored reasoning systems are necessary. We will comment in this subsection on theorem proving, aiming at proving mathematical theorems and on knowledge representation. This idea can be seen as going back to the ideas of Gottfried Wilhelm Leibnitz (1646 - 1716), who already at this time had a dream of formalisation, and even automatisisation of mathematics.

0/2

0/2/0



Go Back

View Size

Single Page

Continuous

Theorem Proving

A recent success is the proof of Robbins conjecture, which even reached the New York Times. For details see [McCunes homepage](#).

In 1933, E. V. Huntington presented the following basis for Boolean algebra:

$$x + y = y + x.$$

[commutativity]

$$(x + y) + z = x + (y + z).$$

[associativity]

$$n(n(x) + y) + n(n(x) + n(y)) = x.$$

[Huntington equation]

Shortly thereafter, Herbert Robbins conjectured that the Huntington equation can be replaced with a simpler one:

$$n(n(x + y) + n(x + n(y))) = x.$$

[Robbins equation]

Robbins and Huntington could not find a proof, and the problem was later studied by Tarski and his students.

The proof that solves the Robbins problem was found October 10, 1996, by the theorem prover EQP. EQP is similar in many ways to the more well known program [Otter](#). The main differences are that EQP has associative-commutative (AC) unification, is restricted to equational logic, and offers more paramodulation strategies. See the [EQP preprint](#) for details.

See also:



[Go Back](#)

[View Size](#)

[Single Page](#)

[Continuous](#)

McCune

Otter

EQP preprin

0/2/2

Knowledge Representation In many Artificial Intelligence the representation and manipulation of knowledge is a central task. To this end numerous graphic-oriented formalisms have been invented. In figure ?? a small example is given.

An informal semantics of this graphical notation states, that both, *man* and *animal* are *mamal* and that *man* have a *nationality* and an *age*, whereas an *animal* has *age* and no *nationality*.

A closer investigation of the semantics of such a formalism would show, that this is nothing than a pictorial representation of the following set of predicate logic formulae:

$$\forall x(man(x) \Rightarrow mammal(x))$$

$$\forall x(animal(x) \Rightarrow mammal(x))$$

$$\forall x(man(x) \Rightarrow \exists y age(x, y))$$

$$\forall x(man(x) \Rightarrow \exists y nationality(x, y))$$

$$\forall x(mammal(x) \Rightarrow \exists y age(x))$$

See also:

Gellrich/Gellrich: Mathematik (1) - Schaltalgebra

0/2/3

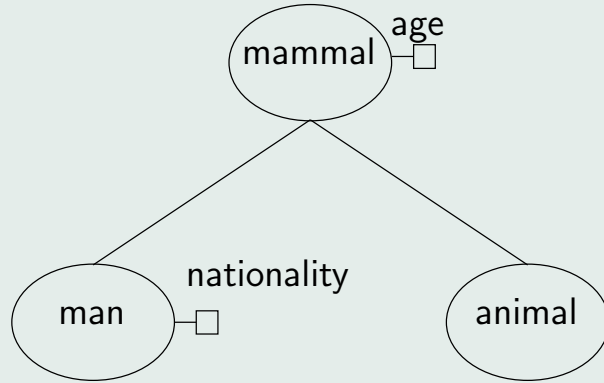
Go Back

View Size

Single Page

Continuous

Example



0/3

0/3/0

Problem 1

In a criminal case the following facts are proved:

1. At least one of the three persons X,Y,Z is guilty.
2. If X guilty and Y are innocent, then Z is guilty.

These circumstances are not sufficient to accuse one of them but it can be said for certain that at least one of two persons must be guilty. Which two are these?

0/3/1

Problem 2

Which of the following syllogisms are valid? Give a reason for your answer or give a counterexample.

1. All M are P , some S are not M then: Some S are P .
2. All M are P , some S are not M then: Some S are not P



Go Back

View Size

Single Page

Continuous

3. All P are M , some S are not M then: Some S are not P .

0/3/2

Problem 3

In a meeting there are 100 politicians discussing with each other. Everyone of them is either corruptibly or uncorruptibly. Following facts are known:

1. at least a politician is uncorruptibly.
2. In each case of two politicians is at least one corruptibly .

How Many of the politicians are corruptibly, how many uncorruptibly?

0/3/3

Problem 4

The anthropologist Abercrombie entered the island of the knights and the mucker with a slack feeling he have never had before. He knew that very wondrous people lived on this island: The knights made only true propositions the mucker false propositions every time. Abercrombie knew also that he had to find a friend before he could experience something. He had to find someone whose propositions he can trust. So he asked the first three people of the island he met to find a knight. Aberbrombie asked Arthur at first: Are Bernard and Charles both knights? Arthur answered: Yes they are! Abercrombie asked then: Is Bernard a knight? With a big surprise he get the answer: No! Is Charles a knight or a mucker?(Raymond Smullyan)

Deduce your answer and give a reason for it.

0/3/4

Problem 5

A little island had exact 100 inhabitants. Every inhabitant said always the truth or lied always either. One Researcher came one day on the island and questioned the inhabitants sequentially. The first told: "There is at least a liar with us." The second said: "At least two liars live among us." etc.. The last finally claimed: "There are 100 liars on this Island." How many liars were there



Go Back

View Size

Single Page

Continuous

really?

The researcher went to another island with 99 Inhabitants a year later. In an interview the inhabitants of these island spoke completely corresponding like on the first island, i.e. the n th inhabitants said: "There are at least n liars here." What can you say about this island?

0/3/5

Problem 6

In a village the priest explained one Sunday: "It was confessed to me that there are men who are unfaithful in our village. However, the confessing secret forbids it to me to call the names. It will you nevertheless learn all of them, if we proceed as follows: Any woman who for certain knows that her husband is unfaithful shall throw him in the following night out of the house."

However, the problem was that every woman knows all about every other husband but nothing about her husband. In the next morning the priest went through the streets; not one single man was turned adrift. Also on the next day he saw nobody. But at 100th acre he saw men, who had been thrown out of the house by their wives. How many?

0/3/6

Problem 7

There is a hotel with countable infinitely many rooms $0, 1, 2, \dots$. All rooms are vacant. Now, a ω -decker bus comes with ω many seats on every deck. How can all of the passengers be accommodated in the hotel?



Go Back

View Size

Single Page

Continuous

Layout

Format document for print



Go Back

View Size

Single Page

Continuous